Unified Service Platform for Accessing Grid Resources

Shaochong Feng
Dept. 2, Mechanical Engineering College, Shijiazhuang, China
Email: fscsat@gmail.com

Yuanchang Zhu and Yanqiang Di
Dept. 2, Mechanical Engineering College, Shijiazhuang, China
Email: {YuanchangZ, YanDi}@gmail.com

***Abstract*—Web Services Resource Framework (WSRF) redefines Grid Services standards and extends Web Services by adding stateful resources. Using GT4 to develop WSRF Grid Services is a taxing work, and it is difficult to build and deploy these services dynamically. Addressing these issues, this paper proposes a unified service platform which can provide a series of unified service interfaces for accessing kinds of different Grid resources. On the platform, Grid resources are independent of the service interfaces. The platform provides unified service interfaces to access different Grid resources on server, so Grid services developers only pay attentions to realizing the native methods of Grid resources and configuring necessary resource database. The remainder work of composing typical Grid Services such as mapping the resources into the service interfaces would be automatically finished by the platform. It means that Grid Services development becomes native application development. What is more, there are no needs to restart the service container when deploying/undeploying Grid resources, so it does not affect other resources. The platform provides service-users with two types of clients, one is for directly invoking the unified interfaces and the other is a proxy client associating to the specific Grid resource. Finally, the test shows that the service development and deployment is much easier on this platform and the service performs well.**

***Index Terms*—WSRF, GT4, Grid resources, Web Services**

I. INTRODUCTION

Web Services Resource Framework (WSRF) specifies stateful services by adding stateful resources to stateless Web Services. The Globus Toolkit (GT) is a software toolkit can use to program Grid-based applications. The Globus Toolkit 4(GT4), in fact, includes a complete implementation of the WSRF specification. GT4 Java WS Core is the common runtime component provides a set of Java libraries and tools which are needed to build both WS and non-WS services. This paper discusses the issue that using GT4 to build WSRF Grid Services [1] [2].

Developer may encounter some problems using GT4 Java WS Core to developing Grid Services and Grid-based applications.

1) Heavy coding workload to generate a WSRF service. Generating Grid Services Using GT4 is different to developing native applications. Besides the kernel code of the services and resources, many other configuration files, build files and scripts must be finished manually. As Fig.1 shows, writing a simple stateful web service that uses WSRF to keep state information needs at least 4 steps:

- Define the service's interface. This is done with *WSDL*.
- Implement the service. This is done with Java.
- Define the deployment parameters. This is done with WSDD and JNDI.
- Compile everything and generate a GAR file. This is done with Ant.

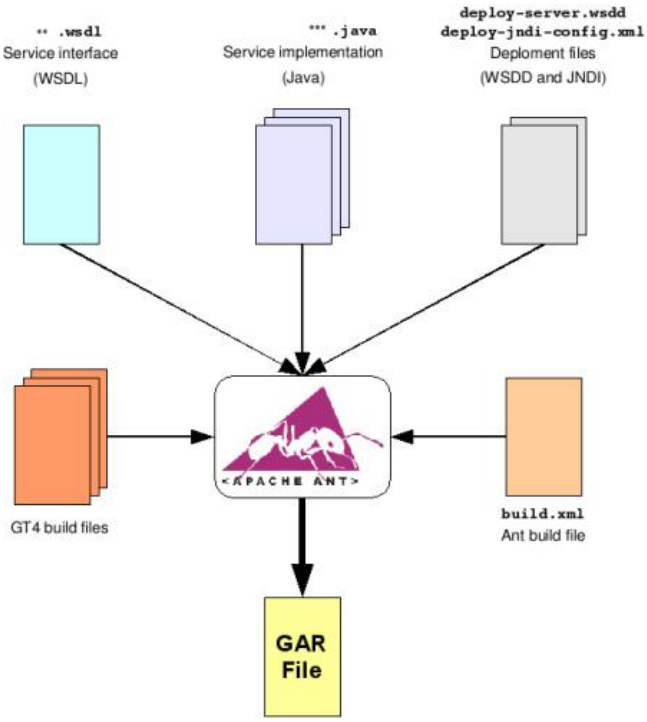

Figure 1. Generating a WSRF service GAR file

It is a great block for the newer, and is a load for the trained programmer.

doi:10.4304/jsw.6.5.930-936

2) Difficult to dynamically deploy and undeploy. The final work of deploy a WSRF Grid Service is placing the generated GAR files to services container. Although GT4.2 supports dynamic deployment in standalone container, the third party Web services containers such as Tomcat are not supported. If developers want to deploy or undeploy Grid Services, they have to stop the services container first, executing the deploying or undeploying, and then restart the services container. The other active stateful services and resources running in the container would be consequentially affected. What is more, services modifying and transferring cannot be achieved dynamically.

Addressing these issues, this paper proposes a Unified Service Platform (USP) for accessing Grid resources based on GT4 Java WS Core. Under USP, the main workload of developing Grid applications is realizing the native methods of the Grid resources, just like developing native application. The Grid resources can be dynamically deployed and undeployed under USP regardless what kind of Web services container is used.

The remainder of the paper is organized as follows: Section II briefly summarizes the related researches and works. Section III analyzes the modes of accessing the Grid resources. Section IV describes the architecture of USP. Section V presents the experiment on USP. Conclusions and future work are presented in Section VI.

II. RELATED WORKS

The research group form University of Marburg proposed GDT (Grid Development Tools) which is part of the Marburg Ad-hoc Grid Environment (MAGE). GDT is a bundle of Eclipse Plugins useful for Service and Application Development, Workflow Creation, Grid Management and others in the Eclipse Integrated Development Environment [5]. GDT greatly reduced the workload under Eclipse to develop Grid applications. In [7], they modified the Axis web service engine utilized by GT4 to allow dynamic loading and unloading of Grid services. Hot Deployment Service (HDS) was constructed to provide applications with the capability to remotely deploy, undeploy and redeploy services onto a running node. In [8] an approach to discover Grid resources and to deploy Grid services based on peer-to-peer technologies was presented.

Reference [9] and reference [10] specially researched the dynamic services deployment. Eun-Kyu Byun developed Universal Factory Service (UFS) that provided a dynamic Grid service deployment mechanism and a resource broker called Door service [9]. Jon B. Weissman proposed a dynamic service architecture consists of several core services and components, the kernel was the Adaptive Grid Service (AGS) [10]. However, these researches are all based on OGSI/GT3, and with the development of Grid technology, OGSI/GT3 has been replaced by WSRF/GT4. The OGSI services don't have states information, so the proposed approaches in [9] and [10] are not compatible with WSRF.

FuQiang Li presented an approach to deploy visualization services dynamically in [11]. The Uniform Visualization Service (UVS) is developed in this paper to provide dynamic visualization services deployment on the Grid, and a service named Agency Service, is used as a resource broker/dispatcher and service states communicator. Service users send requests to Agency Service. The Agency Service find available or applicable services from service center, retrieve the available service codes from the service codes center, and find available resources from MDS. The Agency Service can transfer the service codes to the available resources. The UVS on the resources creates UVS instance to create the service instance with the service codes. The service instance has its own service resources, which can be interacted and collaborated with the service users. In this architecture, the service providers only concern on developing the visualization service codes but not finding available resources.

Addressing the issue that services container can only accommodate language-special services, Pu Liu and Michael J. Lewis described the implementation of an approach that allows Grid services developers to write their code in one language, and have the services running in different service containers, namely GT4, WSRF.NET, and gSOAP and ACE [12].

Li Qi and Doc. Hai Jin proposed a highly available dynamic deployment infrastructure (HAND) in [13], which addressed dynamic service deployment at both the container level and the service level.

These researches are significative for this paper, but all of them concerned the service itself, the service interfaces are tightly bound to the resources. USP proposed in this paper keeps the services interfaces and the resources completely separated, users can access the Grid resources through the same service interface. USP provides the unified service interface and the mechanism of managing different resources on server, so the Grid Service developing is predigested to developing resources and configurating database.

III. MODES OF ACCESSING GRID RESOURCES

WSRF adds stateful resources to stateless Web Services. So the WSRF service interfaces usually bind the resources. The kernel idea of this paper is thoroughly disparting the service interfaces and Grid resources. USP publishes the unified WSRF service interfaces which access Grid resources through "proxy resource". The related terms are listed as follows:

1) Proxy Resource (PR): PR is a Java class, and it is the resource directly related to the published service interfaces by USP. PR implements the defined functions in USP's WSDL file, and provides access to the real resource.

2) Real Resource (RR): RR also is a Java class, and it is the resource with the final functions. RR is reserved in the resource database.

3) Instance of Proxy Resource (IPR): IPR is an object of class PR.

4) Instance of Real Resource (IRR): IRR is an object of class RR.

USP takes the user-defined Grid resources as RR, and publishes unified service interfaces for accessing the RRs. The published service interfaces only retrieve PR, and they know nothing about the RR. USP automatically maps the access from client to PR to the pointed RR's native methods.

When client invokes the service published by USP, the PR and RR on server may interact with each other in three modes:

A. *Singleness IPR mode*

As Fig.2 shows, in the singleness IPR mode, there is only one instance of proxy resource. All clients access IRRs through the Singleness IPR, so the invoked service interface should include the parameters to indicate the invoking RR, the specified IRR and the exact method with all of the method's parameters.

B. *IPR-RR mode*

As Fig.3 shows, in the IPR-RR mode, USP maintains an instance of proxy resource for each real resource class, and one IPR may contain several instances of real resource. The clients access the different instances of the same RR class through the same IPR. It means the invoked RR can be judged by the IPR, so the invoked service interface should include the parameters to indicate the specified IRR and the exact method with all of the method's parameters.

C. *IPR-IRR mode*

As Fig.4 shows, in this mode, instance of proxy resource and instance of real resource are peer to peer. The clients access the IPR-IRR pairs. So the invoked service interface should only include the exact method with method's parameters.

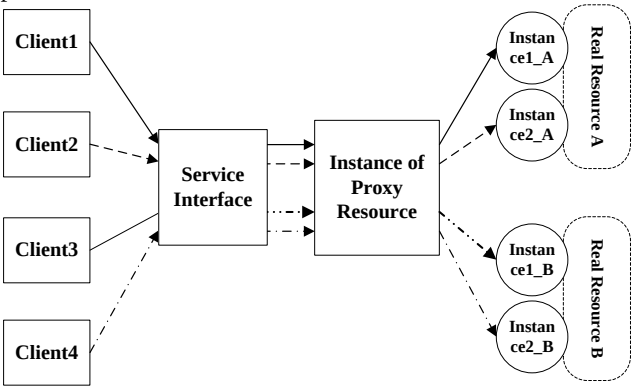

Figure 2. Singleness IPR mode

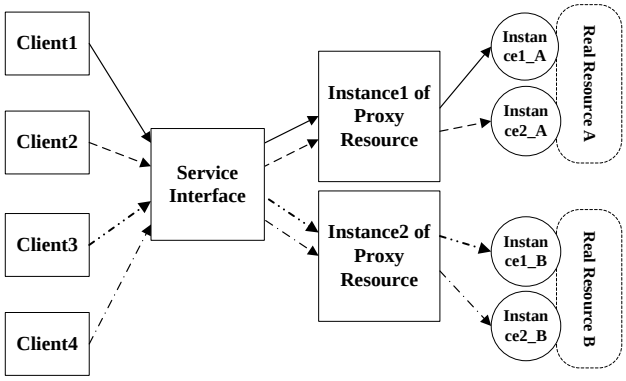

Figure 3. IPR-RR mode

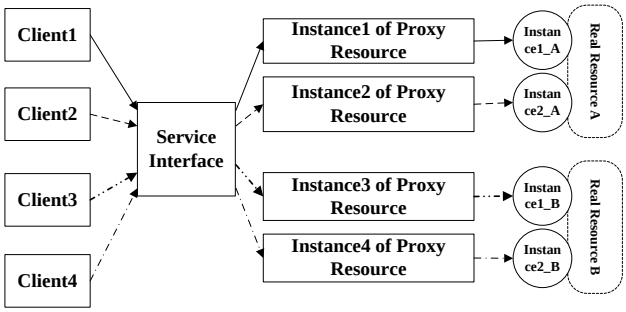

Figure 4. IPR-IRR mode

Obviously, in the first two modes, it is probable that client access different IRRs through the same IPR, and the fault in one IRR accessing may fail several of the other IRRs. What is more, in view of GT4, the IPR and IRR could not be created together with these two modes, extra attentions must be paid to manage the IRR, and it may increase the complexity of the system. Anyway, this paper adopts the IPR-IRR mode.

IV. IMPLEMENT OF THE MAIN COMPONENTS

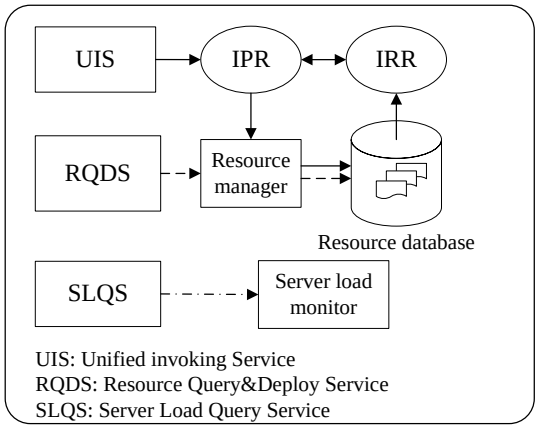

Figure 5. Structure of USP server

Besides IPR and IRR, USP server is composed of three WSRF services and several backup modules like Fig.5 shows.

A. Resource database

Resource database maintains the RR class information, such as the full class name (including class package), public class methods and so on. The information of the active IRR is also recorded here.

B. Resource manager

This module takes responsibility to map IPR to IRR. The functions are listed as following:

- Queries and configures the resource database.
- Obtains the RR class according to the queried RR's name utilizing Java reflection/introspection mechanism. Farther, all of the RR class native methods can be achieved.
- Maps the access to IPR to corresponding IRR according to the necessary parameter provided by the UIS.

C. Server load monitor

The performance parameters such as CPU type, CPU utilization, memory size, memory utilization and bandwidth are gained by server load monitor module.

D. Unified Invoking Service(UIS)

As the kernel part of USP, UIS is a WSRF service, and it provides clients with the unified invocation interfaces to requesting the RR on server. IPR-IRR mode described in Fig.3 is adopted here. The main interfaces published by UIS are listed as follows:

1) Service interface for resource instantiation: The service interface *CreateResource(Sting strRRName)*can create an IPR-IRR pair. The *strRRName* parameter specified the identity of the accessed RR. Receiving the invoking on this interface, server creates an instance of PR, then queries the resource database, obtains the RR by *strRRName*, and creates the object of it. So an IPR-IRR pair is maintained on server for each client.

2) Service interface for accessing the RR: Because the native methods are different between different RRs, the service interface for accessing to RR's native method is designed to be unique, and it is *WSRFFunc(String strParam, String strFuncName)*. The *strFuncName* parameter specifies RR's native method to be accessed whose parameters are included in the *strParam* parameter. Receiving the request on this interface, server invokes IPR, and then maps this invoking to the IRR through resource manager module.

3) Interface for the notification: in GT4, resource exposes a resource property (RP) as a topic for client to subscribing, a notification would be triggered each time the value of the RP changes, and then client subscribing the topic receives the notification. The topic published by RR are also uncertain, so the USP implies a unified methods *WSRFNoty(String str)*in UIS. PR publishes one topic for client subscribing, and all the message of the notification such as the real topic of RR and the value of RP is encoded in the *str* parameter.

E. Resource Query&Deploy Service(RQDS)

RQDS provides WSRF services facilitating users to program under USP.

1) Service interface for querying RR information: the service interface *QueryAllClass()* queries all of the RRs, and generates a text file for client. The text file lists all the class names of RRs.

2) Service interface for querying the pointed RR information: the service interface *QueryClass(String strRRName)* queries detailed information of pointed RR specified by the *strRRNamer* parameter, and generates three file for client. The first file lists all the public methods with parameters of the class. The second file shows an example of invoking the unified service interfaces published by UIS. The third file is a Java file, it is the proxy client, and sometimes a proxy notifier is also generated if the pointed RR can notify the client. UIS publishes only one service interface *WSRFFunc* for clients to access RR's methods. Although client may invoke this interface follow the example in first file, the client program would be very complex, because clients have to make sure the *strFuncName* parameter to specify the RR's method, and encode the *strParam* parameter according to the achieved second text file generated by *QueryClass* interface. So, the *QueryClass* interface can build new Java classes based on the queried RR class and the client stubs, and they are the proxy client and proxy notifier. The proxy client encapsulates the client stubs and provides same public methods with the pointed remote RR; analogously the proxy notifier provides the same interfaces with the pointed remote RR. So if client instantiates the proxy client, and invokes its methods, the proxy client would map the invoking to the *WSRFFunc* request, and finally the request would be sent to access the RR's corresponding method on server. It greatly advantages the USP users.

3) Service interface for deploying Grid resource: the service interface *DeployingRR(String strConfgFile, String strRR)* is for users to remotely deploy the RR and configure the resource database. The *strConfgFile* parameter specifies a local text file, which should define the following elements: RR full class name, location on the server, and dependence relationship. The other *strRR* parameter refers the native Java class file to be deployed as RR. The file transmission from client to server is based on GridFTP.

All of the interfaces of the RQDS are related to the resource manager module which reads and writes the resource database.

F. Server Load Query Service(SLQS)

SLQS provides clients with the server's performance information such as the CPU type, CPU utilization, memory size, memory utilization and bandwidth.

G. Process of accessing the resources

Under USP, the process of accessing the Grid resources is described in Fig.5 (it is supposed that client has invoked

RQDS, generated the proxy client, and instantiated the IPR-IRR pair by calling *CreateResource*).

1) Client invokes the native method *Func(String str)* of the proxy client.

2) Proxy client transforms *Func(String str)* to *WSRFFunc(String strParam, String strFuncName)*, and invoke the client stub. The parameter of *Func* is encoded to be the *strParam* parameter, and proxy client also specifies the invoking method by *strFuncName* parameter.

3) Client stub sends the request *WSRFFunc* to UIS.

4) UIS finds the corresponding IPR of the client.

5) USP invokes the IPR's *WSRFFunc* method.

6) 7) Resource manager queries resource database and fine the IRR's *Func* method according to the *strFuncName* parameter.

8) UIS decodes the *strParam* parameter, and call the *Func* method.

Finally, client receives the responds from server.

The implement of the notification is a converse process.

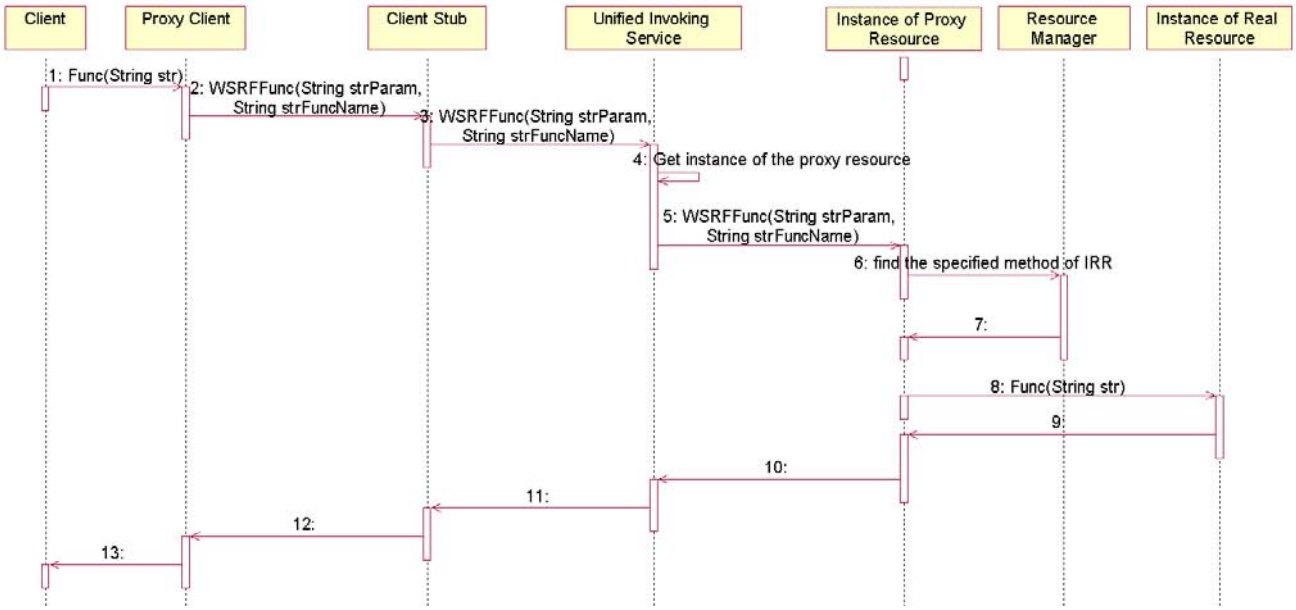

Figure 6. Consequent diagram of accessing Grid resources based on USP

V. PERFORMANCE TEST

The proposed unified service platform is based on GT4 Java WS Core, it provides unified WSRF service interfaces to accessing the resources on server through the IPR-IRR pair, and reduces the workload in service development and configuration. USP adds middleware to the interaction between service interfaces and the real resources, so the service performance would be affected consequentially.

The performance test is executed to examine the performance lost and development efficiency increase. The test mainly inspected the response time to finish a service invoking. What is more, we compared the workload by measuring time consumption for a trained GT4 programmer to developing and deploying new Grid Service.

A. Response time test

We took a simple RR class *ClassA* with native method *int Func(String str)* for example. The *Func* directly returns an integer value, and does nothing else.

Using GT4, the *ClassA* is instantiated in hard code by resource factory when Creating IPR, and service interface directly calls IRR's *Func* method and get the returned value. Under USP, the *ClassA* was recorded in resource database, USP queries the database, gets *ClassA* and its *Func* method utilizing Java reflection/introspection mechanism, and then invokes its method. It is necessary to code and decode the parameters in the interaction between IPR and instance of *ClassA*.

The testing environment is described in Tab.1. We separately carried out experiments in Windows and Linux, and all the tests were executed in LAN connected by a switch.

References [14] and [15] have mentioned that in Windows environment, the service response time seemed to be affiliated with the length of the parameters. If the data length is about between 200 bytes and 3000 bytes, the latency is relatively small, as long as out of this range, longer or shorter, the latency would be much heavier. So in Windows test, the length of *str* parameter was set to 500 bytes.

In Windows, we separately executed 50 tests under USP and GT4. In each test, we continually invoked the service interface for 100 times, measured the invoking response time and in the end of each test, the mean of all the time consumption values were calculated as the result of this test.

The test scheme in Linux was similar except that the length of *str* parameter was 5 bytes.

TABLE I. TEST ENVIRONMENT

server	CPU	Pentium(R)4 2.6G
	memory	512M
	Operation system	WinXP pro sp2
		Ubuntu8.04
	Grid toolkit	GT4.0.8
	JDK	Java-1.5.0
	Service container	Tomcat5.5.29
	Ethernet	100M
client	CPU	Pentium(R)4 2.6G
	memory	512M
	Operation system	WinXP pro sp2
		Ubuntu 8.04
	JDK	Java-1.5.0
	Service container	Tomcat5.5.29
	Ethernet	100M
other	switch	DLink 100M

Fig.7 shows the test data in Windows, while Fig.8 indicates the test in Linux. Tab.2 analyses the data. From these two figures, we can see that the values of service response time under USP and GT4 are approximate. Tab.2 shows that under USP, the mean service time in Windows rises from 19.27ms to 20.34ms, and the one in Linux rises from 37.84ms to 38.56ms comparing to the service directly utilizing GT4. The service performance loss is no more than 6%.

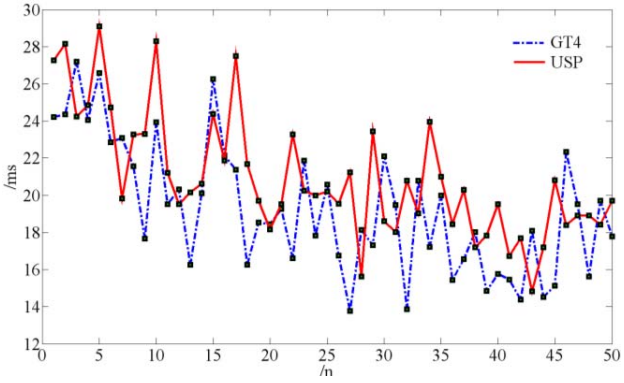

Figure 7. Service time consumption in Windows

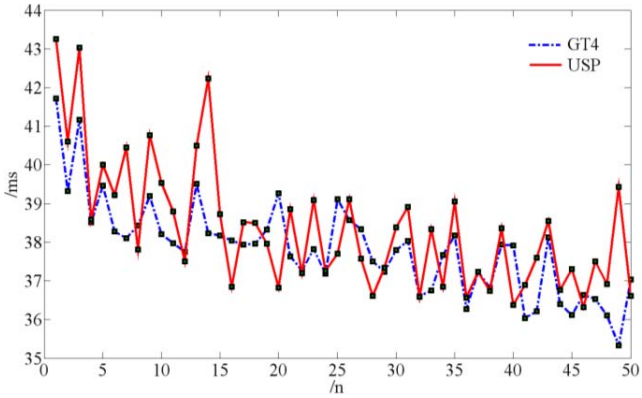

Figure 8. Service time consumption in Linux

TABLE II. DATA IN WINDOWS

	Windows(ms)			Linux(ms)		
	max	min	mean	max	min	mean
GT4	17.2	13.77	19.27	41.72	35.34	37.84
USP	29.09	14.82	20.34	43.26	36.32	38.56

In fact, the performance of GT4 service is not stable for some reasons, just like what Fig.7 and Fig.8 have showed, the service response time always floats in a certain range. So a little service time rising is not meaningful to Grid Services.

So, the negative effect on service performance taken by USP is not evident.

B. *Workload test*

The performance can be ensured, how about the workload? We separately measured the time consumption for a trained programmer to finish new Grid Service utilizing GT4, GDT and USP. The function of Grid Service is same to the one used in response time test. The comparing of time consumption is listed in Tab.3. We can see that the developing efficiency increases a lot.

TABLE III. TIME CONSUMING COMPARE

Tools	USP	GDT	GT4
Time	4.2 min	11.4min	37.8 min

C. *Test conclusion*

From the test, we can easily come to the conclusion that Grid Service development under USP is efficient and the outcome service performs almost as well as the service directly generated by GT4.

VI. CONCLUSION

A unified service platform for accessing Grid recourses is proposed in the paper. USP separates the service development and resource development, and provides

unified service interfaces to access different resources. The main workload for a USP user willing to deploy Grid resources is realizing the native methods and configuring the resource database. The resources under USP can be deloyed/undeployed dynamically. Besides the common client stub which can directly invoke the unified service published by USP, users can also achieve the proxy client through resource query&deploy service. The proxy client and proxy notifier could greatly facilitate the service users. The final experiment proofs that the service performance under USP is acceptable for Grid applications and development workload reduce a lot.

This paper only proposes a prototype. The future works include management of the third-party resources used by RR, GUI for USP users, high QoS of USP, and so on.

ACKNOWLEDGMENT

The authors wish to thank Dr. Xianguo Meng from Dept.2 of Mechanical Engineering College.

REFERENCE

[1] http://www.globus.org/toolkit/.(2010-10-11)

[2] Borja Sotomayor. The Globus Toolkit4 Programmer's Tutorial. http://gdp.globus.org/gt4-tutorial/.(2010-2-5)

[3] http://tomcat.apache.org. (2010-2-5)

[4] Ken Arnold, Jams Gosling, David Holmes. The Java™ Programming Language, Fourth Edition. Pearson Education, Inc. 2006

[5] http://mage.uni-marburg.de/trac/gdt/wiki. (2009-12-3)

[6] T. Friese, M. Smith, and B. Freisleben. GDT:A Toolkit for Grid Service Development. In Proc. of the 3rd International Conference on Grid Service Engineering and Management, pages 131-148, 2006

[7] M. Smith, T. Friese, and B. Freisleben. Intra-Engine Service Security for Grids Based on WSRF. In Proceedings of the 2005 IEEE International Symposium on Cluster Computing and Grid (CCGRID'05), pages 644-653, 2005.

[8] Kay Dornemann and Bernd Freisleben. Discovering Grid Resources and Deploying Grid Services Using Peer-to-Peer Technologies. In Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops, pages 292-297, 2009

[9] Eun-Kyu Byunt, Jae-Wan Jangt, Wook Jungt et al. A Dynamic Grid Services Deployment Mechanism for On-Demand Resource Provisioning. In Proceedings of the 2005 IEEE International Symposium on Cluster Computing and the Grid, pages 863-870, 2005

[10] Jon B. Weissman, Seonho Kim, and Darin England, Supporting the Grid Service Dynamic Lifecycle. In Proceedings of the 2005 IEEE International on Cluster Symposium and Computing the Grid, pages 808- 815,2005

[11] Fu Qiang Li, Bin Gong, Cheng Xing. Dynamic Visualization Service Deployment in Grid Scientific Workflow. In Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing, pages 201-205, 2008

[12] Pu Liu, Michael J. Lewis. Unified Dynamic Deployment of Web and Grid Services. In the Proceedings of 2007 IEEE Conference on Web Service, pages 26-34, 2007

[13] L. Qi, H. Jin, I. Foster, and J. Gawor. HAND: Highly Available Dynamic Deployment Infrastructure for Globus Toolkit 4. In the Proceedings of 15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing, pages155-162, 2007

[14] Shaochong Feng, Yanqiang Di, Yuanchang Zhu, et al. Developing WSRF-Based Web Service RTI Using GT4. In Proceedings of 2009 First International Workshop on Education Technology and Computer Science, pages 1066-1069, 2009

[15] Feriese,Thomas."Service-Oriented Ad Hoc Grid Computing", [Doctoral dissertation], Fachbereich Mathmatik und Informatik Universit Marburg ,2006

Shaochong Feng was born in Hebei, China. He receives his Master's degree in Guidance, Navigation and Control from Mechanical Engineering College in 2007. His technical interests include distributed modeling and simulation, Grid computing and Cloud computing.

Yuanchang Zhu was born in Heilongjiang, China. He received the B.A., M.A., and Ph.D in 1982, 1988 and 2005. He is a professor at Mechanical Engineering College. His study interests include M&S, Cloud Computing and so on.

Yanqiang Di was born in Hebei, China. He received the B.A., M.A., and Ph.D in 1995, 1998 and 2009. He is a teacher at Mechanical Engineering College. His study interests include M&S, Grid Computing and database system.